

Why would you consider a Scripting Language as JavaScript as your Backend Platform?

Rigtig mange bruger allerede JavaScript i deres frontend og det er en fordel at bruge det samme sprog i hele stakken. Samtidig er JavaScript et af de mest brugte sprog og det stiger stadig i popularitet. Node.js kan teoretisk set have 4000 forbindelser åbnet samtidig hvis man har 8gb ram på sin pc.

Explain Pros & Cons in using Node.js + Express to implement your Backend compared to a strategy using, for example, Java/JAX-RS/Tomcat

Som nævnt ovenfor giver det mening at bruge node.js i sin backend hvis man allerede bruger den til sin frontend. Dette sørger for at man ofte kan genbruge kode fra frontend til backend og omvendt. En anden fordel ved at bruge node.js er npm - Node Package Manager, der er omkring 350.000 værktøjer og bliver tilføjet hver uge.

Selv om der er mange fordele ved at bruge node.js, møder det stadig stor modstand blandt mange, her er nogle af årsagerne til dette: Callbackhell, et af de helt store problemer med javascript er callbackhell hvor flere metoder bliver kaldt efter hinanden, dette kan man dog helt eller delvist undgå ved brug af promises eller async/await. Selv om JavaScript og dermed også Node.js er asynkron og non blocking kan der stadig opstå "kø" i event loopet hvis man laver CPU intensive opgaver, dette kan medføre en forsinkelse hos slutbrugeren. Npm med dets store udvalg af værktøjer kan både være en fordel og en ulempe, mange af værktøjerne kan være af lav kvalitet og kan derfor skabe større problemer end de løser. Der er intet kvalitets eller sikkerhedstjek så det handler om at være vågen når man skal finde det rigtige værktøj til jobbet.

Node.js uses a Single Threaded Non-blocking strategy to handle asynchronous task. Explain strategies to implement a Node.js based server architecture that still could take advantage of a multi-core Server.

Fordi Node.js er single-threaded kan den ikke bruge flere threads som man ser i f.x. Java. I stedet skal/kan man bruge "clustering" gennem f.x. Node clustering api'et hvor man "forker" processerne

<https://dzone.com/articles/multicore-programming-in-nodejs>

Explain briefly how to deploy a Node/Express application including how to solve the following deployment problems:


- **Ensure that you Node-process restarts after a (potential) exception that closed the application**

For at være sikker på at node genstarter skal man bruge en process manager. Denne sørger for at genstarte ens app hvis den crasher, viser memory forbrug og CPU forbrug og viser performance, som et ekstra lag sikkerhed i tilfælde af servercrashes

kan man også bruge et init system som genstarter processmanageren når OS genstarter

- **Ensure that you Node-process restarts after a server (Ubuntu) restart**

Se ovenstående

-  **Ensure that you can take advantage of a multi-core system**
- **Ensure that you can run “many” node-applications on a single droplet on the same port (80)**

Til dette kan man bruge `app.use()`. Man skal også bruge en reverse proxy som f.eks. nginx.

<https://itnext.io/hosting-multiple-apps-on-the-same-server-implement-a-reverse-proxy-with-node-a4e213497345>

Explain the difference between “Debug outputs” and application logging. What’s wrong with `console.log(..)` statements in our backend-code.

`Console.log` statements er ikke asynkrone og der vil derfor være ventetid hver gang man printer noget.

Demonstrate a system using application logging and “coloured” debug statements.

Explain, using relevant examples, concepts related to testing a REST-API using Node/JavaScript + relevant packages

Vi kan bruge Mocha og Chai til at teste vores JavaScript backend

Explain, using relevant examples, the Express concept; middleware.

Middelware er funktioner der kører efter et request bliver sendt men før der kommer et response tilbage, dette kan sørge for at hashe et password eller verificere en bruger før han/hun kan tilgå en side. `//Third-party middleware`

```
app.use(bodyParser.urlencoded())
//Custom middleware
app.use(function (req, res, next) {
  console.log('Time:', Date.now()+"Log all requests");
  next();
});
//Route functions
app.all('/somePath',function(req,res){
  console.log("Log on all request for /somePath (GET,POST, PUT, DELETE)")
})
app.get('/somePath', function(req, res){
  res.send('This is a route');
});
app.post('/somePath', function(req, res){
```

```
//.. Do something with the request parameters
});
//Built-in middleware (the only one left in V4.x.x)
app.use(express.static('./public'));
```

Compare the express strategy toward (server side) templating with the one you used with Java on second semester.

Den måde man embedder kode minder meget om hinanden i EJS og JSP:

SP tags:

- <%! Variable declaration or method definition %>
- <%= Java valid expression %>
- <% Pure Java code %>

EJS tags:

- <% 'Scriptlet' tag, for control-flow, no output
- <%= 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
- <%= Outputs the value into the template (HTML escaped)
- <%- Outputs the unescaped value into the template
- <%# Comment tag, no execution, no output
- <%% Outputs a literal '<%'
- %> Plain ending tag
- -%> Trim-mode ('newline slurp') tag, trims following newline
- _%> 'Whitespace Slurping' ending tag, removes all whitespace after it

Demonstrate a simple Server Side Rendering example using a technology of your own choice (pug, EJS, ..).

<https://gist.github.com/MortenBB/341ec76200bca3ce4770733907ca522c>

Explain, using relevant examples, your strategy for implementing a REST-API with Node/Express and show how you can "test" all the four CRUD operations programmatically using, for example, the *Request* package.

Rest:

<https://gist.github.com/MortenBB/3c0b91ed806025427f3394609f6be5ba>

Mocha og Chai tests:

<https://gist.github.com/MortenBB/452798c9b8898f512b427bc0ef1e27bc>

Explain, using relevant examples, about testing JavaScript code, relevant packages (Mocha etc.) and how to test asynchronous code.

Testværktøjer som Mocha og Chai gør det let at teste JavaScript. Mocha gør det muligt at skrive unit-tests næsten som vi kender dem fra Java, man starter dem med "it" og en kort beskrivelse af hvad testen gør.

Med Chai kan vi tilføje en expect hvilket gør det nemmere at læse testene. Nu kan man skrive testene sådan: `to.be.equal("res")`.

Asynkron kode testes meget lige som man kender det fra `async await` se nedenfor:

 **Explain, using relevant examples, different ways to mock out databases, HTTP-request etc.**

For at lave et fake request kan man bruge Nock

Explain, preferably using an example, how you have deployed your node/Express applications, and which of the Express Production best practices you have followed.

NoSQL, MongoDB and Mongoose

Explain, generally, what is meant by a NoSQL database.

En NoSQL database har ikke, modsat SQL-databaser, skemaer der dikterer hvad de forskellige modeller indeholder.

Explain Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a traditional Relational SQL Database like MySQL.

"In most situations, SQL databases are vertically scalable, which means that you can increase the load on a single server by increasing things like CPU, RAM or SSD. NoSQL databases, on the other hand, are horizontally scalable. This means that you handle more traffic by sharding, or adding more servers in your NoSQL database." -

<https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2>

Hvis man skal lave en stor database hvor man ofte ændrer i strukturen vil en NoSQL database være at foretrække fordi man har mulighed for at ændre enkelte steder uden at det går ud over de tidligere dokumenter. Samtidig har man mulighed for at lave et dokument uden at lave et skema.

En af fordelene ved en SQL-database som MySQL er, at programmet har så mange år på bagen. Det betyder at næsten alle funktioner er blevet testet intensivt, den er tilgængelig på alle styresystemer og virker med alle sprog

Explain reasons to add a layer like Mongoose, on top on of a schema-less database like MongoDB

Mongoose gør det muligt at bruge skemaer i vores database, dette sørger for at vores data er ens samtidig med at det gør det en hel del lettere at skrive databasen

Demonstrate, using a REST-API you have designed, how to perform all CRUD operations on a MongoDB

<https://gist.github.com/MortenBB/bc2d516433b474b89a6ff1dd6955df9e>

Explain the benefits of using Mongoose, and demonstrate, using your own code, an example involving all CRUD operations

Mongoose introducere mange nye ting der ikke findes i MongoDB, blandt disse er f.eks. Middleware et værktøj vi har arbejdet med før hvilket gør det muligt at lave avancerede funktion før data fra databasen bliver sendt eller modtaget. Mongoose implementere også nogle af de features vi kender fra en traditionel database som f.eks. Skemaer, dette gør at det er nemmere hvis man kommer fra en SQL database

<https://stackoverflow.com/questions/18531696/why-do-we-need-what-advantages-to-use-mongoose>

Explain the “6 Rules of Thumb: Your Guide Through the Rainbow” as to how and when you would use normalization vs. denormalization.

- **One:** favor embedding unless there is a compelling reason not to
- **Two:** needing to access an object on its own is a compelling reason not to embed it
- **Three:** Arrays should not grow without bound. If there are more than a couple of hundred documents on the “many” side, don’t embed them; if there are more than a few thousand documents on the “many” side, don’t use an array of ObjectID references. High-cardinality arrays are a compelling reason not to embed.
- **Four:** Don’t be afraid of application-level joins: if you index correctly and use the projection specifier (as shown in part 2) then application-level joins are barely more expensive than server-side joins in a relational database.
- **Five:** Consider the write/read ratio when denormalizing. A field that will mostly be read and only seldom updated is a good candidate for denormalization: if you denormalize a field that is updated frequently then the extra work of finding and updating all the instances is likely to overwhelm the savings that you get from denormalizing.
- **Six:** As always with MongoDB, how you model your data depends – entirely – on your particular application’s data access patterns. You want to structure your data to match the ways that your application queries and updates it.

<https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part->

3

Demonstrate, using your own code-samples, decisions you have made regarding → normalization vs denormalization

Kommer senere (se miniprojekt)

Explain, using a relevant example, a full JavaScript backend including relevant test cases to test the REST-API (not on the production database)

Kommer senere (se miniprojekt)